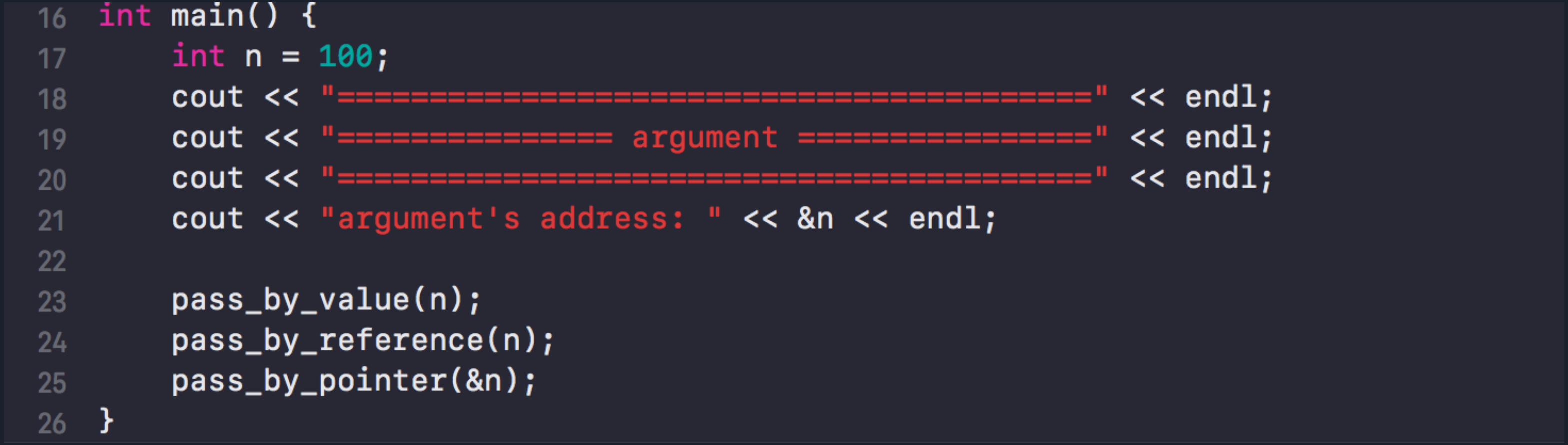
COMP 345 Week 3

Haotao Lai (Eric)
h_lai@encs.concordia.ca
http://laihaotao.me/ta

Parameter-Passing

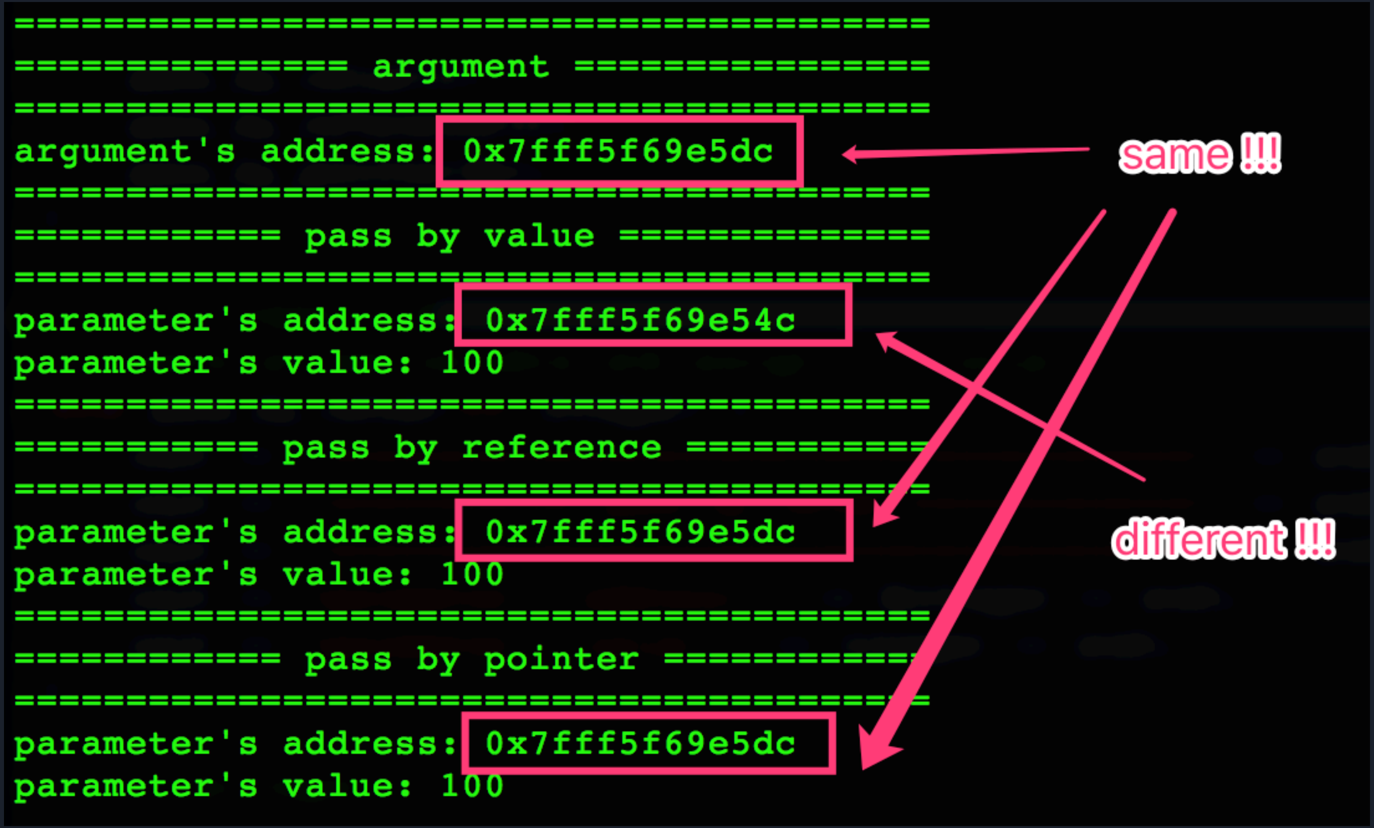
Parameter-Passing

- pass by value: copy the value, and pass the new copied value;
- pass by reference: create a new alias for that parameter and pass the alias;
- pass by pointer: get the address of the parameter and pass that address;

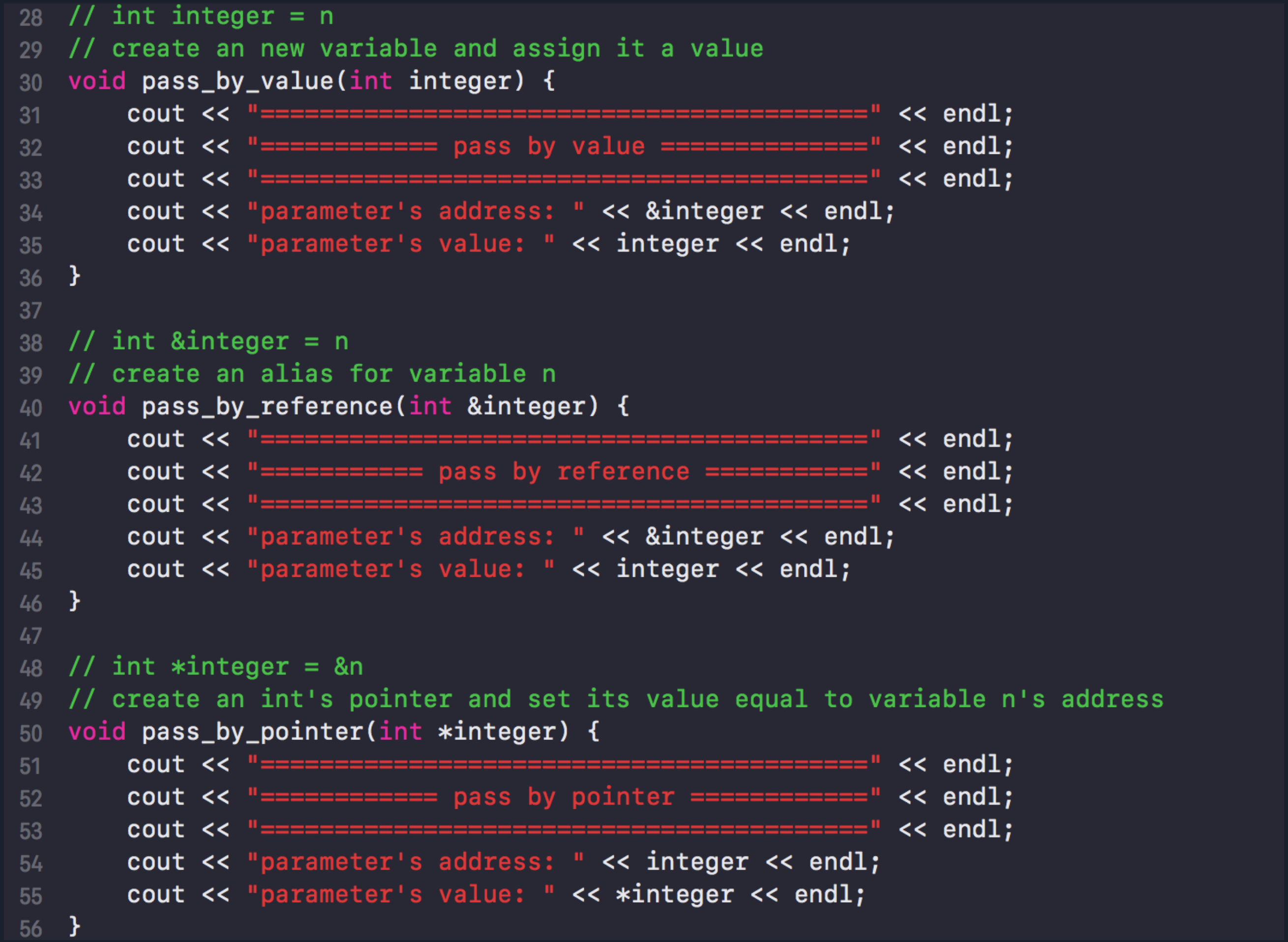


```cpp
int main() {
    int n = 100;
    cout << "=========================================" << endl;
    cout << "=============== argument ================" << endl;
    cout << "=========================================" << endl;
    cout << "argument's address: " << &n << endl;

    pass_by_value(n);
    pass_by_reference(n);
    pass_by_pointer(&n);
}
```



```
=========================================
=============== argument ================
=========================================
argument's address: 0x7fff5f69e5dc
=========================================
============ pass by value ==============
=========================================
parameter's address: 0x7fff5f69e54c
parameter's value: 100
=========================================
=========== pass by reference ===========
=========================================
parameter's address: 0x7fff5f69e5dc
parameter's value: 100
=========================================
============ pass by pointer ============
=========================================
parameter's address: 0x7fff5f69e5dc
parameter's value: 100
```

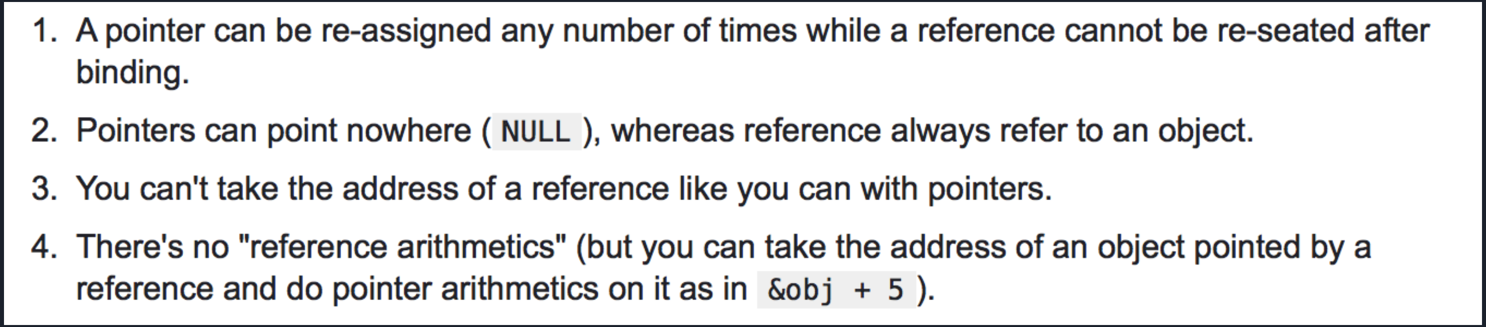


```cpp
// int integer = n
// create an new variable and assign it a value
void pass_by_value(int integer) {
    cout << "=========================================" << endl;
    cout << "============ pass by value ==============" << endl;
    cout << "=========================================" << endl;
    cout << "parameter's address: " << &integer << endl;
    cout << "parameter's value: " << integer << endl;
}

// int &integer = n
// create an alias for variable n
void pass_by_reference(int &integer) {
    cout << "=========================================" << endl;
    cout << "=========== pass by reference ===========" << endl;
    cout << "=========================================" << endl;
    cout << "parameter's address: " << &integer << endl;
    cout << "parameter's value: " << integer << endl;
}

// int *integer = &n
// create an int's pointer and set its value equal to variable n's address
void pass_by_pointer(int *integer) {
    cout << "=========================================" << endl;
    cout << "============ pass by pointer ============" << endl;
    cout << "=========================================" << endl;
    cout << "parameter's address: " << integer << endl;
    cout << "parameter's value: " << *integer << endl;
}
```

Difference between reference and pointer



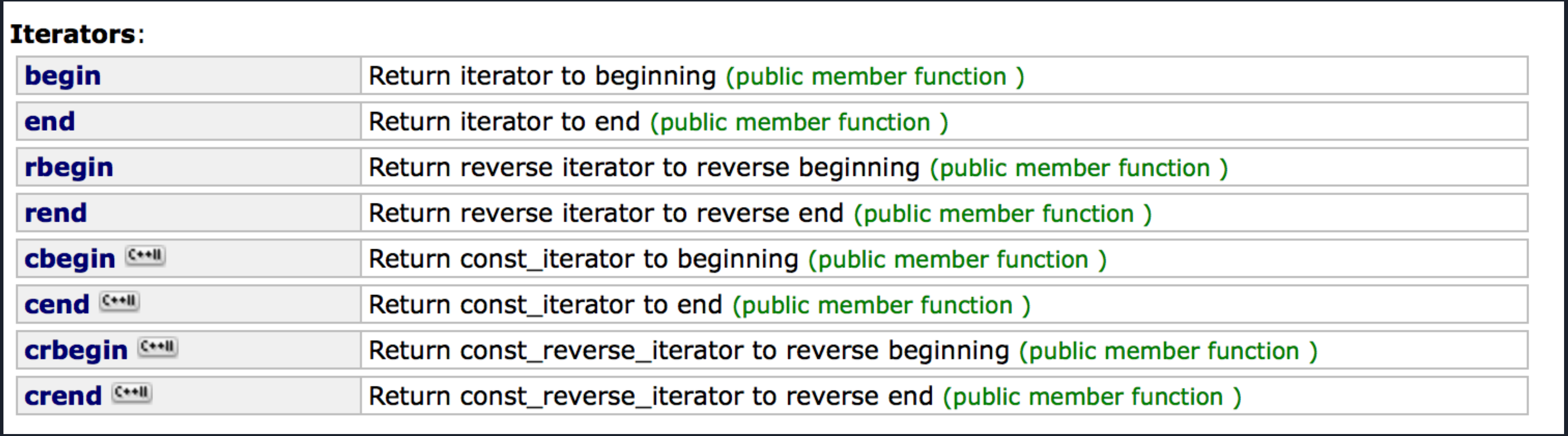
1. A pointer can be re-assigned any number of times while a reference cannot be re-seated after binding.
2. Pointers can point nowhere (`NULL`), whereas reference always refer to an object.
3. You can't take the address of a reference like you can with pointers.
4. There's no "reference arithmetics" (but you can take the address of an object pointed by a reference and do pointer arithmetics on it as in `&obj + 5`).

—— from stackoverflow, know more click here

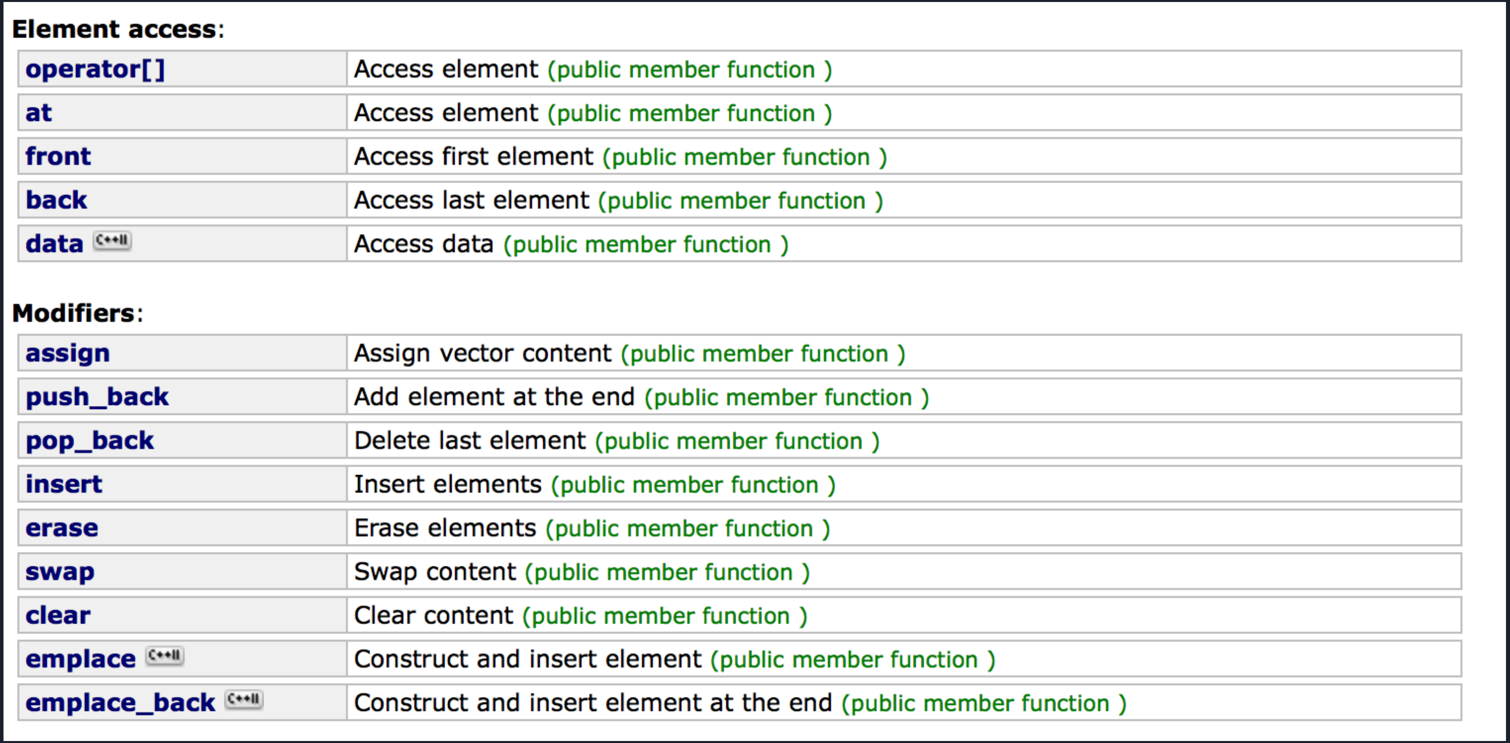
Vector

Vector

- vector<T> in cpp likes List<T> in Java
- Vectors are sequence containers representing arrays that can change in size.
- know more about vector, go here


Iterators:

begin	Return iterator to beginning (public member function)
end	Return iterator to end (public member function)
rbegin	Return reverse iterator to reverse beginning (public member function)
rend	Return reverse iterator to reverse end (public member function)
cbegin C++11	Return const_iterator to beginning (public member function)
cend C++11	Return const_iterator to end (public member function)
crbegin C++11	Return const_reverse_iterator to reverse beginning (public member function)
crend C++11	Return const_reverse_iterator to reverse end (public member function)

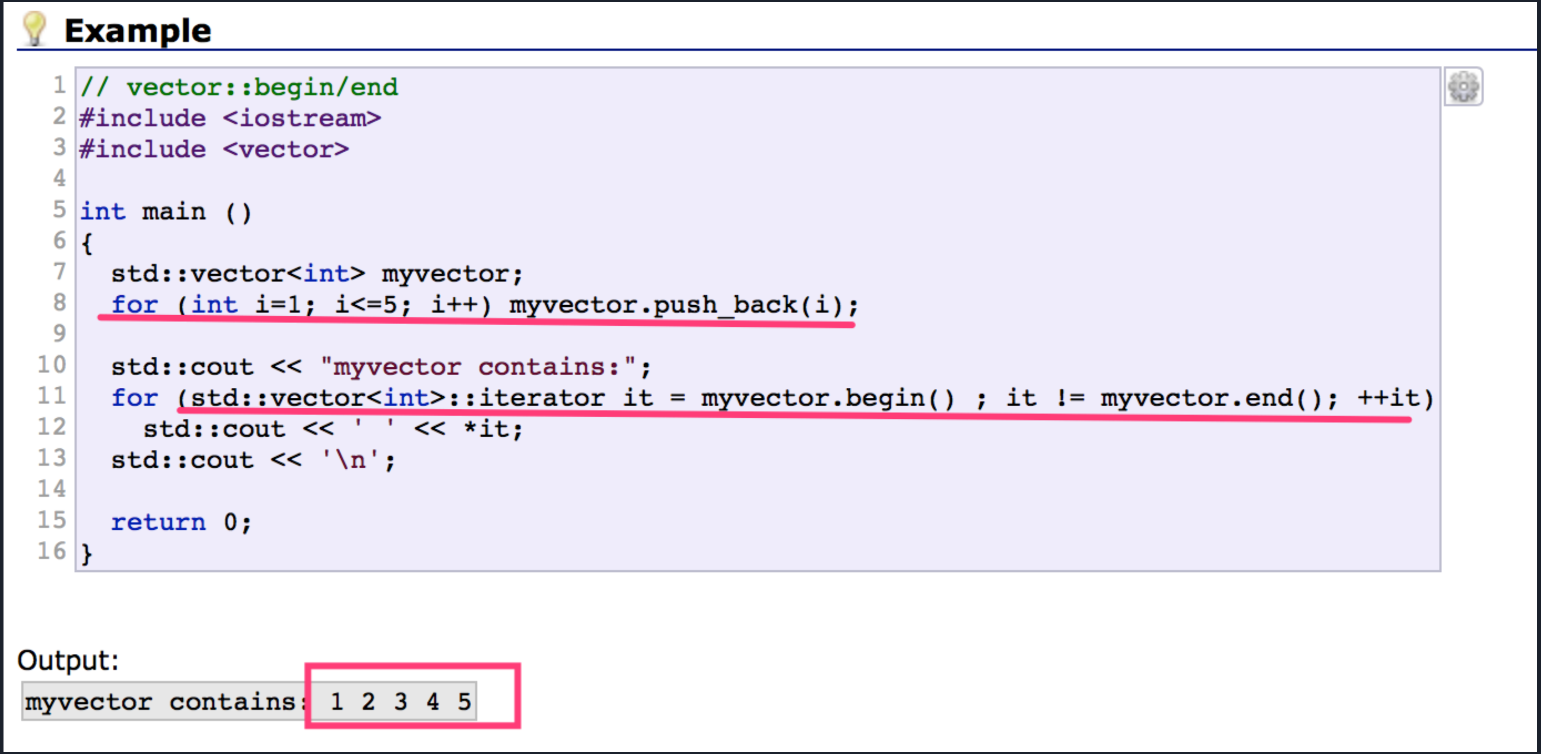


Element access:

operator[]	Access element (public member function)
at	Access element (public member function)
front	Access first element (public member function)
back	Access last element (public member function)
data C++11	Access data (public member function)

Modifiers:

assign	Assign vector content (public member function)
push_back	Add element at the end (public member function)
pop_back	Delete last element (public member function)
insert	Insert elements (public member function)
erase	Erase elements (public member function)
swap	Swap content (public member function)
clear	Clear content (public member function)
emplace C++11	Construct and insert element (public member function)
emplace_back C++11	Construct and insert element at the end (public member function)



Example

```
// vector::begin/end
#include <iostream>
#include <vector>

int main ()
{
  std::vector<int> myvector;
  for (int i=1; i<=5; i++) myvector.push_back(i);

  std::cout << "myvector contains:";
  for (std::vector<int>::iterator it = myvector.begin() ; it != myvector.end(); ++it)
    std::cout << ' ' << *it;
  std::cout << '\n';

  return 0;
}
```

Output:

```
myvector contains: 1 2 3 4 5
```

Command Line Compile

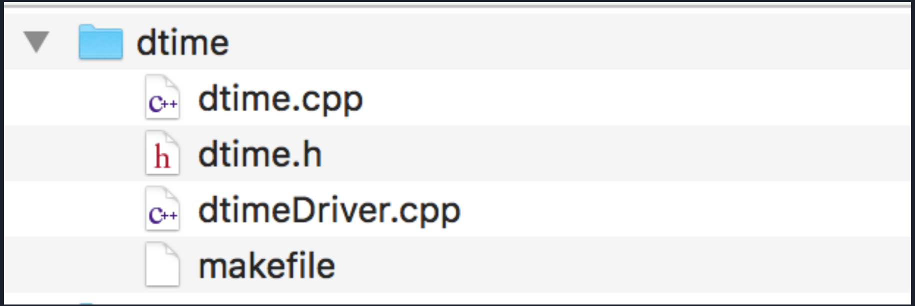
Command Line Compile

- Assume we have two classes: Student.cpp Student.h (Data) and StudentDriver.cpp (Entry)
- cd source_directory
- g++ -c Student.cpp
- g++ -c StudentDriver.cpp
- g++ -o StudentExample Student.o StudentDriver.o
- ./StudentExample

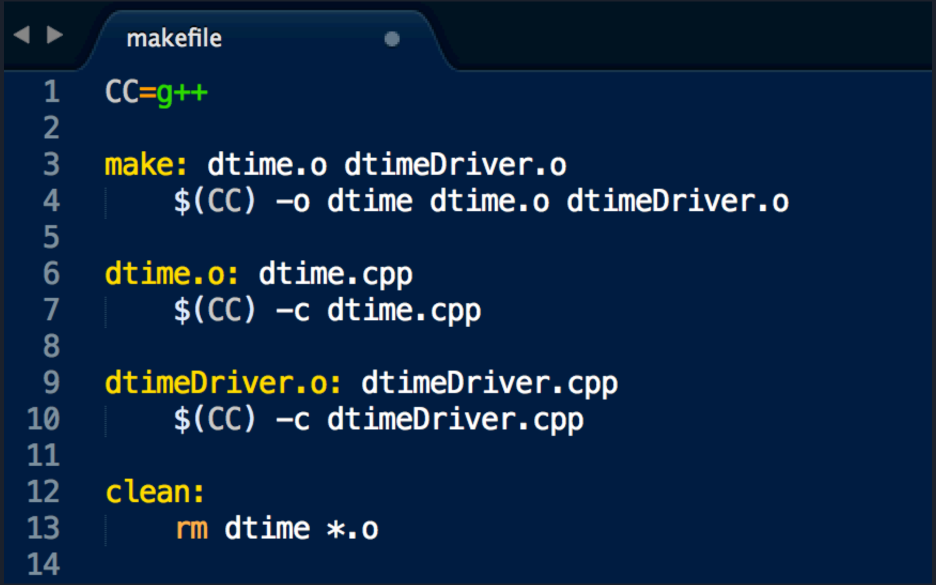
Makefile

Makefile

- if you are working on a large project with 1000+ files
- of course you will compiler them one by one in command line
- you need to write a makefile
- make
- you will get your executable file
- want to know more about make and makefile, click here



```
dtime
    dtime.cpp
    dtime.h
    dtimeDriver.cpp
    makefile
```



makefile

```
CC=g++

make: dtime.o dtimeDriver.o
	$(CC) -o dtime dtime.o dtimeDriver.o

dtime.o: dtime.cpp
	$(CC) -c dtime.cpp

dtimeDriver.o: dtimeDriver.cpp
	$(CC) -c dtimeDriver.cpp

clean:
	rm dtime *.o

```